IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	Siddhartha Nandi, Abhay Kumar Singh, Oleg Kiselev		
Assignee:	VERITAS Operating Corporation		
Title:	System And Method For Dynamically Loadable Storage Device I/O Policy Modules		
Serial No.:	10/717,037	Filing Date:	November 19, 2003
Examiner:	Unassigned	Group Art Unit:	2122
Docket No.:	VRT0091US		

Austin, Texas
April 20, 2004

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

PETITION TO MAKE SPECIAL UNDER 37 CFR §1.102(d)

Dear Sir:

The applicants hereby petition pursuant to 37 CFR §1.102(d) and MPEP § 708.02(VIII) to make the above-identified application special. Please charge Deposit Account No. 502306 for the fee of $130.00 for this petition as set forth in 37 CFR §1.17(h).

Should the Office determine that all the claims presented are not obviously directed to a single invention, the applicants will make an election without traverse as a prerequisite to the grant of special status.

The applicants respectfully submit that a pre-examination search has been performed by a professional search firm in the following classes/subclasses:

Class	Subclasses
G06F	11/20, 11/34, 13/14, 13/16
714	4, 6
H04L	12/56

Enclosed are copies of the following references which are presently believed to be, from among those made of record in the accompanying Information Disclosure

Statement and any previously filed Information Disclosure Statement, the most closely related to the subject matter encompassed by the claims:

US 6,145,028 US 5,944,838 US 2001/0050915
EP 0788055
"Highly Available Storage: Multipathing and the Microsoft MPIO Driver Architecture White Paper," Microsoft Corporation, October 2003

Detailed Discussion of the References

U.S. Patent 6,145,028 (Shank) discloses a hardware independent system and method for adaptively managing and controlling multipath access to an array of data storage devices. Referring to Figure 1, Shank teaches that computer 102 implements a virtual disk driver 134, which translates application 132 I/O requests into I/O virtual disk requests, and also configures the storage device array 104 to implement one or more EMPATH virtual disks 104. The virtual disk driver 134 assigns each of the I/O paths a state of either active (path is available for use in sending I/Os) or passive (an I/O path in the standby mode). At least one of the I/O paths is assigned an active status when the disk is configured. See, e.g., column 4, lines 18-26.

To allow the virtual disk driver 134 to operate with a wide range of storage device array 104 hardware and software implementations, the virtual disk driver 134 comprises a core virtual disk driver 136 and one or more subordinate disk drivers 138. The core virtual disk driver comprises instructions which translate application 132 I/O requests into I/O virtual disk requests. Subordinate disk drivers 138 comprise a series of instructions which implement functionality appropriate for each particular disk array 104 configuration, protocol, and/or hardware implementation. For example, when a storage device array-specific function is required to perform controller switchovers or I/O load balancing, the appropriate subordinate disk driver 138 provides that functionality. In this way, the virtual disk driver 134 provides a generic, hardware-independent solution that supports storage device arrays with open systems. This allows, for example, that the storage device array 104 may be replaced or substituted with one from a different manufacturer, without altering the core virtual disk driver 136. See, e.g., column 4, line 53 through column 5, line 4.

Additionally, Figure 9 illustrates steps performed in one embodiment of Shank's invention incorporating a switch table. First, an inquiry command is sent from the virtual disk driver 134 to the storage array devices 104. This is illustrated in block 902. Next, inquiry data is received 904 from the storage array devices 104. Then, the switch table 140 is scanned 906 to identify a subordinate disk driver 138 corresponding to the storage

array device identification. Finally, the core virtual disk driver 136 is associated 908 with the subordinate disk driver 138. See, e.g., column 6, lines 48-57.

Although Shank teaches the use of a virtual disk driver 134 including a core virtual disk driver 136 and one or more subordinate disk drivers 138, Shank neither teaches nor suggests: (1) receiving a request to load a device policy module into a memory; (2) loading the device policy module into the memory; and (3) informing the device driver of availability of the device policy module, all as required by claim 1. Accordingly, the applicants respectfully submit that claim 1 is allowable over Shank.

U.S. Patent No. 5,944,838 (Jantz) purports to address the problem of providing fast restart of failover of I/O requests from a failed (bad) I/O path to an alternate, operational (good) I/O path, and in a manner that is portable so as to be easily implemented within any host system. See, e.g., column 3, lines 35-40.

In particular, Jantz teaches that a redundant storage control module (also referred to as RDAC or multi-active controller) maintains a queue of pending I/O requests sent for processing via a first asynchronously operating I/O path. In the event of failure of the first asynchronously operating I/O path, the controller restarts the entire queue of pending I/O requests to a second I/O path without waiting for each request to individually fail from the first path. This technique is advantages because some prior techniques required the RDAC module to await failure of each I/O request sent to the failed first I/O path before restarting each failed request on the secondary I/O path, thereby extending the total time required to restart all operations sent to a failed I/O path. Other known techniques provide non-standard features in the lower level driver modules to permit the higher level RDAC modules to directly manipulate dispatch queues maintained for each I/O path within the low level device drivers. See, e.g., the Abstract, column 5, lines 29-63, and column 6, line 57 through column 7, line 24.

Thus, Jantz's RDAC and associated queue neither teach nor suggest: (1) receiving a request to load a device policy module into a memory; (2) loading the device policy module into the memory; and (3) informing the device driver of availability of the device

policy module, all as required by claim 1. Accordingly, the applicants respectfully submit that claim 1 is allowable over Jantz.

EP Application 0788055 (Hodges) discloses a data processing system having multiple independent paths for communication between multiple independent storage controllers, and particularly techniques for efficient management of the queues in a multiple independent path storage subsystem where the requests for accessing the storage devices can be carried out without the need for the queues to be in sync with each other.

Referring to Figure 5, Hodges shows array controller 350 in communication with storage devices 345 and storage controller 320. Array controller 350 comprises two identical independently operating controller paths 352 and 354. Controller paths 352 and 354 together provide for enhanced performance and fault tolerant operation. Array controller 350 is in communication with storage controller 320 over a plurality of communication links 380. Array controller 350 includes a plurality of storage interface adapters 400 (four shown in this example, two in each controller path). Storage interface adapters 400 receive commands (instructions, requests) for access to storage devices 345 from storage controller 320 and initially determine if a request issued by storage controller 320 can be satisfied from either buffer 416 or buffer 418. Buffers 416 and 418 store data received from the devices and the storage controller. If the request can be satisfied from the content of either buffer, that information is communicated back to storage controller 320 via one of the storage interface adapters 400. For each request that cannot be satisfied from the content of one of the buffers, one of the storage interface adapters generates a unique request identifier (also referred to as a label) and sends the request with its unique identifier to array managers 412 and 414. Array managers 412 and 414, in turn, add the request and its label to their respective queues 413 and 415. See, e.g., column 8, lines 8-36.

Array managers 412 and 414 are generally responsible for managing data handling between storage controller 320 and storage devices 345. They further translate data format between storage devices 345 format (native format) and the emulated device format (the format as seen by storage controller 320). Each array manager further

maintains its own task queue. For example, array manager 412 maintains array task queue 413 and array manager 414 maintains array task queue 415. Each array manager updates its own array task queue on the basis of messages received from any one of the storage interface adapters 400 and based on the content of the mailbox in each storage device. Array managers 412 and 414 further manage dispatching device operations to storage devices 345 via device interface adapters 440 and 442, respectively. See, e.g., column 8, lines 37-58.

Thus, while Hodges teaches a multipath I/O system using queues that do not need to be synchronized with each other, Hodges neither teaches nor suggests: (1) receiving a request to load a device policy module into a memory; (2) loading the device policy module into the memory; and (3) informing the device driver of availability of the device policy module, all as required by claim 1. Accordingly, the applicants respectfully submit that claim 1 is allowable over Hodges.

U.S. Patent Application 2001/0050915 (O'Hare) purports to address problems in multipath, multihop systems. In some data storage device arrangements, a first data storage device may be connected to a second data storage device and a processor may only be able to send commands to the second data storage device indirectly using the first data storage device. In other words, the processor only has a direct connection to the first data storage device, and an indirect connection to the second data storage device. If the processor wishes to instruct the second data storage device to perform a data operation, the processor may use a remote system call using the first data storage device. The processor may issue a remote procedure call to the first data storage device which instructs the second data storage device to perform a data operation, for example. This situation can be extended to one or more additional levels of indirect data storage devices. Such indirect communication may require instructions to pass through an indirect connection of multiple data storage devices. There is no way for the processor to, for example, instruct a third data storage device using system calls or remote procedure calls due to the additional storage device layering in the data storage device arrangement. Generally, similar problems may exist in data storage device arrangements

that also include more than the foregoing three levels of indirect data storage it device access from a processor. See, e.g., paragraphs 0007-0008.

To address this problem, O'Hare discloses a computer system for providing multihop system calls. Data storage devices are interconnected and also connected to one or more hosts. Each data storage device classifies a data operation as a system call, a remote system call, or a multihop system call. Also described is a multipath multihop system call in which one or more communication paths may be selected using predetermined and/or dynamic communication path selection techniques. The number of communication paths determined may be in accordance with parameters that are included in a multipath multihop system call, tunable system parameters, or a combination of the two. See, e.g., the Abstract.

While describing techniques for implementing multihop communication, O'Hare neither teaches nor suggests: (1) receiving a request to load a device policy module into a memory; (2) loading the device policy module into the memory; and (3) informing the device driver of availability of the device policy module, all as required by claim 1. Accordingly, the applicants respectfully submit that claim 1 is allowable over O'Hare.

"Highly Available Storage: Multipathing and the Microsoft MPIO Driver Architecture White Paper", Microsoft Corporation, October 2003 (Microsoft) describes the Microsoft MPIO (multipath input/output) solution in Windows 2000 Server, Windows Server 2003, and Windows Storage Server 2003. MPIO is designed to work in conjunction with device specific modules (DSMs) written by vendors. Microsoft provides a sample device-specific module, or DSM, designed to provide a software interface between the multipath driver package and the hardware device. Vendors must adapt this generic DSM to the specifics of their device or devices. This joint solution allows vendors to design hardware solutions that are tightly integrated with the Windows operating system, and also enables Microsoft to correctly accommodate the non-generic characteristics of each vendor's storage device (such as whether there are multiple active controllers or the controllers have only standby capability), without having to design the MPIO solution in anticipation of each possible difference. See, e.g., page 6, bottom.

The Windows operating system relies on the Plug and Play (PnP) Manager to dynamically detect and configure hardware (such as adapters or disks), including hardware used for high availability/high performance multipathing solutions. The Microsoft MPIO driver development kit is designed to work seamlessly with PnP architecture. The Microsoft MPIO software supports the ability to transparently balance I/O workload, without administrator intervention. MPIO determines which paths to a device are in an active state and can be used for load balancing. Each vendor's load balancing policy (which may use any of several algorithms, such as round robin, the path with the fewest outstanding commands, or a vendor unique algorithm) is set in the DSM. This policy determines how the I/O requests are actually routed. If the DSM returns a path that is inactive, the failover process is initiated. The MPIO driver package, in combination with the vendor DSM, supports end-to-end path failover. The process of detecting failed paths and recovering from the failure, like load balancing, is automatic, extremely fast, and completely transparent to the IT organization. See, e.g., page 8, top, and page 9.

The MPIO driver package consists of three multipath drivers: the port filter driver, the disk-driver replacement, and the bus driver; all of which are implemented in the kernel mode of the operating system. The MPIO driver package works in combination with both the PnP Manager, the disk class driver, the port driver, the miniport driver, and each vendor's device-specific module (DSM) to provide full multipathing functionality. See, e.g., page 11 ("MPIO Drivers").

Although Microsoft teaches a variety of multipath driver features, Microsoft neither teaches nor suggests: (1) receiving a request to load a device policy module into a memory; (2) loading the device policy module into the memory; and (3) informing the device driver of availability of the device policy module, all as required by claim 1. Accordingly, the applicants respectfully submit that claim 1 is allowable over Microsoft.

Conclusion

In summary, the applicants respectfully submit that none of the references located during the pre-examination search, or otherwise made of record by the applicants, teaches or suggests (at least): (1) receiving a request to load a device policy module into a memory; (2) loading the device policy module into the memory; and (3) informing the device driver of availability of the device policy module, all as required by claim 1.

Accordingly, the applicants respectfully request that this petition be granted, and that the present application receive expedited examination. Should any issues remain that might be subject to resolution through a telephonic interview, the Office is requested to telephone the undersigned.

Express Mail Label No:
EV 304738713 US

Respectfully submitted,

Marc R. Ascolese
Attorney for Applicant(s)
Reg. No. 42,268
512-439-5085
512-439-5099 (fax)